



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/708,021	02/03/2004	Jonas Hogstrom	BORL/0217.00	2020
28653	7590	01/26/2007	EXAMINER	
JOHN A. SMART 708 BLOSSOM HILL RD., #201 LOS GATOS, CA 95032			CHEN, QING	
			ART UNIT	PAPER NUMBER
			2191	
SHORTENED STATUTORY PERIOD OF RESPONSE		MAIL DATE	DELIVERY MODE	
3 MONTHS		01/26/2007	PAPER	

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/708,021	HOGSTROM ET AL.	
	Examiner Qing Chen	Art Unit 2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) Responsive to communication(s) filed on 03 February 2004.
 2a) This action is FINAL. 2b) This action is non-final.
 3) Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) Claim(s) 1-63 is/are pending in the application.
 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
 5) Claim(s) _____ is/are allowed.
 6) Claim(s) 1-63 is/are rejected.
 7) Claim(s) _____ is/are objected to.
 8) Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) The specification is objected to by the Examiner.
 10) The drawing(s) filed on 03 February 2004 is/are: a) accepted or b) objected to by the Examiner.
 Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
 Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
 11) The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

- 12) Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 a) All b) Some * c) None of:
 1. Certified copies of the priority documents have been received.
 2. Certified copies of the priority documents have been received in Application No. _____.
 3. Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

- | | |
|---|---|
| 1) <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) | 4) <input type="checkbox"/> Interview Summary (PTO-413) |
| 2) <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | Paper No(s)/Mail Date. _____ |
| 3) <input checked="" type="checkbox"/> Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date <u>20060530, 20060530</u> . | 5) <input type="checkbox"/> Notice of Informal Patent Application |
| | 6) <input type="checkbox"/> Other: _____ |

DETAILED ACTION

1. This is the initial Office action based on the application filed on February 3, 2004.
2. **Claims 1-63** are pending.

Oath/Declaration

3. The oath or declaration is defective. A new oath or declaration in compliance with 37 CFR 1.67(a) identifying this application by application number and filing date is required. See MPEP §§ 602.01 and 602.02.

The oath or declaration is defective because:

- It does not state that the person making the oath or declaration believes the named inventor or inventors to be the joint inventors of the subject matter which is claimed and for which a patent is sought.
- It is not dated by the named inventors.

Specification

4. The disclosure is objected to because of the following informalities:
 - The specification contains the following typographical errors:
 - A hyphen (-) should be added between “Hewlett” and “Packard” on page 18, paragraph [0040].
 - The reference number “410” for indicating “compiler” should be changed to “420” on page 78, paragraph [0163], since reference number “420” is used to designate “compiler” in Figure 4.

Art Unit: 2191

- o The phrase "... code element that that model element represents ..." should presumably read "... code element that the model element represents ..." on page 80, paragraph [0166].

Appropriate correction is required.

5. The disclosure is objected to because any computer program listing may, and any computer program listing having over 300 lines (up to 72 characters per line) must, be submitted on a compact disc in compliance with § 1.52(e). A compact disc containing such a computer program listing is to be referred to as a "computer program listing appendix." The "computer program listing appendix" will not be part of the printed patent. The specification must include a reference to the "computer program listing appendix" at the location indicated in § 1.77(b)(5).

6. The disclosure is objected to because it contains embedded hyperlinks and/or other form of browser-executable code on page 6, paragraph [0012] and page 14, paragraph [0033].
Applicant is required to delete the embedded hyperlinks and/or other form of browser-executable code. See MPEP § 608.01.

7. The use of trademarks, such as IBM, MICROSOFT, WINDOWS, BORLAND, DELPHI, MACINTOSH, LINUX, SOLARIS, UNIX, FREEBSD, INTEL, PENTIUM, HP LASERJET, BLUETOOTH, 3COM, DELL, APPLE, SUN, SOLARIS, SUN MICROSYSTEMS, JAVA, and JBUILDER, has been noted in this application. Trademarks should be capitalized wherever they appear (capitalize each letter OR accompany each trademark with an appropriate designation

symbol, *e.g.*, ™ or ®) and be accompanied by the generic terminology (use trademarks as adjectives modifying a descriptive noun, *e.g.*, “the JAVA programming language”).

Although the use of trademarks is permissible in patent applications, the proprietary nature of the marks should be respected and every effort made to prevent their use in any manner, which might adversely affect their validity as trademarks.

Claim Objections

8. **Claims 1-63** are objected to because of the following informalities:

- **Claims 1-63** contain a typographical error: the claim marker (*e.g.*, [c1], [c2], etc.) for each claim should be deleted.
- **Claims 9, 31, and 50** contain a typographical error: the word “technique” should be changed to plural form, since more than one traversal technique is recited in the claims.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

9. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

10. **Claims 4, 17-20, 26, 39, 40-42, 45, 46, and 58-61** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Art Unit: 2191

Claims 4, 26, and 45 contain the trademark or trade name JAVA. When a trademark or trade name is used in a claim as a limitation to identify or describe a particular material or product, the claim does not comply with the requirements of the 35 U.S.C. 112, second paragraph. *Ex parte Simpson*, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the trademark or trade name cannot be used properly to identify any particular material or product. A trademark or trade name is used to identify a source of goods, and not the goods themselves. Thus, the use of a trademark or trade name in a claim to identify or describe a material or product (in the present case, a specific programming language) would not only render a claim indefinite, but would also constitute an improper use of the trademark or trade name.

Claim 46 depends on Claim 45 and, therefore, suffers the same deficiency as Claim 45.

Claims 17, 18, 20, 39, 40, 42, 58, 59, and 61 recite the limitation “some information.” The term “some” is a relative term, which renders the claims indefinite. The term “some” is not defined by the claims nor does the specification provide a standard for ascertaining the requisite degree and one of ordinary skill in the art would not be able to reasonably determine the scope of the invention. In the interest of compact prosecution, the Examiner subsequently does not give any patentable weight to this limitation for the purpose of further examination.

Claim 19 depends on Claim 18 and, therefore, suffers the same deficiency as Claim 18.

Claim 41 depends on Claim 40 and, therefore, suffers the same deficiency as Claim 40.

Claim 60 depends on Claim 59 and, therefore, suffers the same deficiency as Claim 59.

Claim Rejections - 35 USC § 101

11. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

12. **Claims 22-63** are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 22 and 63 recite “a downloadable set of processor-executable instructions.” The downloadable set of processor-executable instructions are not claimed as embodied on a computer-readable medium that would allow the processor to execute the instructions. Therefore, the claims are directed to functional descriptive material *per se*, and hence non-statutory.

Claims 23-42 are directed to systems. However, the recited components of the systems appear to lack the necessary physical components (hardware) to constitute a machine or manufacture under § 101. Therefore, these claim limitations can be reasonably interpreted as computer program modules—software *per se*. Furthermore, the specification discloses that the corresponding apparatus element may be configured in software or firmware (*see Page 15, Paragraph [0034]*). Therefore, the claims are directed to systems of functional descriptive material *per se*, and hence non-statutory.

The claims constitute computer programs representing computer listings *per se*. Such descriptions or expressions of the programs are not physical “things.” They are neither computer components nor statutory processes, as they are not “acts” being performed. Such claimed

computer programs do not define any structural and functional interrelationships between the computer program and other claimed elements of a computer, which permit the computer program's functionality to be realized. In contrast, a claimed computer-readable medium encoded with a computer program is a computer element, which defines structural and functional interrelationships between the computer program and the rest of the computer, that permits the computer program's functionality to be realized, and is thus statutory. See *Lowry*, 32 F.3d at 1583-84, 32 USPQ2d at 1035.

The result of **Claims 43-63** is directed to the act of "making the reconstructed model available," which does not appear to be a tangible result so as to constitute a practical application of the idea. The act of "making the reconstructed model available" is merely a thought or an abstract idea and does not appear to produce a tangible result even if the step does occur, since the result of that step is not conveyed in the real world.

Therefore, the claims do not meet the statutory requirement of 35 U.S.C. § 101, since the claims are not directed to a practical application of the § 101 judicial exception producing a result tied to the physical world.

Claim Rejections - 35 USC § 102

13. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

14. **Claims 1-6, 12, 15-28, 34, 37, 39-47, 53, and 56-63** are rejected under 35 U.S.C. 102(b) as being anticipated by Goodwin et al. (US 6,199,195).

As per **Claim 1**, Goodwin et al. disclose:

- creating a model describing business objects and rules of the application (*see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML). "*);
- creating source code for the application, including representing the model within the source code itself (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... "*);
- compiling the source code into an executable application (*see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422*

(*.*idl*), which are processed by an *idl-to-Java compiler*, e.g., *Visibroker's IDL2JAVA*, for the generation of *CORBA ORB services* ... ");

- running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application (see Column 15: 45-65, "The data server 332 operates at run time ... " and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services."); and
- while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework (see Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes." and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions." and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases.").

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the model comprises a Unified Modeling Language (UML) model (see Column 6: 37-44, "... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).").

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

Art Unit: 2191

- wherein the source code is created using a programming language (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... "*).

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is a selected one of JavaTM, Pascal, and C# (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... "*).

As per **Claim 5**, the rejection of **Claim 3** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that JavaTM supports the reflection technique.*).

As per **Claim 6**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application (*see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data*

Art Unit: 2191

server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314. ").

As per **Claim 12**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- after reconstructing the model at run-time, testing integrity of the reconstructed model (*see Column 10: 7-11, "Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ... ".*)

As per **Claim 15**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework (*see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ... ".*)

As per **Claim 16**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the model is initially created using a modeling tool, and wherein the source code is compiled using a compiler (*see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database*

308.”; Column 14: 34-40, “Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker’s IDL2JAVA, for the generation of CORBA ORB services ... ”).

As per **Claim 17**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in source code as language constructs (see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ” It is inherent that the Java™ files represent some information of the model in source code as language constructs.).

As per **Claim 18**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in source code as attributes (see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ” It is inherent that the Java™ files represent some information of the model in source code as attributes.).

As per **Claim 19**, the rejection of **Claim 18** is incorporated; and Goodwin et al. further disclose:

Art Unit: 2191

- wherein attributes comprise specifiers to structural code elements (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files contain specifiers to structural code elements.*).

As per **Claim 20**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in code artifacts that exist expressly for carrying model information in source code (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.*).

As per **Claim 21**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- a computer-readable medium having processor-executable instructions for performing the method of claim 1 (*see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ... "*).

As per **Claim 22**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- a downloadable set of processor-executable instructions for performing the method of claim 1 (*see Column 9: 52-57, "The code generator 330 writes source code objects to a*

directory and returns a uniform resource locator (URL) identified to a client application 338. An operator of the client application (338) is then required to go to the location identified by the uniform resource locator and download the generated code. ").

As per **Claim 23**, Goodwin et al. disclose:

- a modeling tool for creating a model describing business objects and rules of the application (*see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML). "*);
- a module for creating source code for the application and representing the model within the source code itself (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... "*);
- a compiler for compiling the source code into an executable application (*see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... "*); and
- a run-time framework that is able to reconstruct the model from the executable application and use it for providing services (*see Column 15: 45-65, "The data server 332*

operates at run time ... ” and “When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services. ”).

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the model comprises a Unified Modeling Language (UML) model (*see Column 6: 37-44, “... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML). ”*).

As per **Claim 25**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the source code is created using a programming language (*see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ”*).

As per **Claim 26**, the rejection of **Claim 25** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is a selected one of Java™, Pascal, and C# (*see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ”*).

Art Unit: 2191

As per **Claim 27**, the rejection of **Claim 25** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that Java™ supports the reflection technique.*).

As per **Claim 28**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application (*see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314. ".*).

As per **Claim 34**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- a submodule for testing integrity of the reconstructed model (*see Column 10: 7-11, "Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ... ".*).

As per **Claim 37**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework (*see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ... "*).

As per **Claim 39**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent some information of the model in source code as language constructs (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that the Java™ files represent some information of the model in source code as language constructs.*).

As per **Claim 40**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent some information of the model in source code as attributes (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that the Java™ files represent some information of the model in source code as attributes.*).

As per **Claim 41**, the rejection of **Claim 40** is incorporated; and Goodwin et al. further disclose:

- wherein attributes comprise specifiers to structural code elements (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files contain specifiers to structural code elements.*).

As per **Claim 42**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent some information of the model in code artifacts that exist expressly for carrying model information in source code (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.*).

As per **Claim 43**, Goodwin et al. disclose:

- creating a model for developing an application using Unified Modeling Language (UML) technique (*see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML). "*);

Art Unit: 2191

- generating source code to implement the model (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... "*);
- amending the source code for storing model information in the source code (*see Column 13: 65-67 through Column 14: 1-9, "Output from the code generator 330 can be combined with other user defined codes to create an application. "*);
- compiling the amended source code into an executable application and running the executable application on the computer system (*see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... "; Column 15: 45-65, "The data server 332 operates at run time ... " and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services. ");*
- reconstructing the model from the executable application (*see Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes. " and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions. " and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases. "); and*
- making the reconstructed model available for supporting operation of the executable application (*see Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes. " and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions. " and "The data*

server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases. ”).

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the source code is implemented using a programming language (*see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ”*).

As per **Claim 45**, the rejection of **Claim 44** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is a selected one of JavaTM, Pascal, and C# (*see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ”*).

As per **Claim 46**, the rejection of **Claim 45** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model from the executable application (*see Column 13: 52-55, “Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ” It is inherent that JavaTM supports the reflection technique.*).

As per **Claim 47**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed by a run-time framework to provide services to the executable application (*see Column 16: 65-67 through Column 17: 1, “The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314. ”*).

As per **Claim 53**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- after reconstructing the model, testing integrity of the reconstructed model (*see Column 10: 7-11, “Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ... ”*).

As per **Claim 56**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory (*see Column 5: 45-47, “The illustrated system 100 has a processor 102 coupled to a memory 104 ... ”*).

As per **Claim 57**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

Art Unit: 2191

- wherein the model is initially created using a modeling tool, and wherein the amended source code is compiled using a compiler (*see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308. "; Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... ".*).

As per **Claim 58**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in source code as language constructs (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that the Java™ files represent some information of the model in source code as language constructs.*).

As per **Claim 59**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in source code as attributes (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that the Java™ files represent some information of the model in source code as attributes.*).

As per **Claim 60**, the rejection of **Claim 59** is incorporated; and Goodwin et al. further disclose:

- wherein attributes comprise specifiers to structural code elements (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files contain specifiers to structural code elements.*).

As per **Claim 61**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing some information of the model in code artifacts that exist expressly for carrying model information in source code (*see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.*).

As per **Claim 62**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- a computer-readable medium having processor-executable instructions for performing the method of claim 43 (*see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ... "*).

As per **Claim 63**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

Art Unit: 2191

- a downloadable set of processor-executable instructions for performing the method of claim 43 (*see Column 9: 52-57, "The code generator 330 writes source code objects to a directory and returns a uniform resource locator (URL) identified to a client application 338. An operator of the client application (338) is then required to go to the location identified by the uniform resource locator and download the generated code. "*).

Claim Rejections - 35 USC § 103

15. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

16. **Claims 7, 8, 29, 30, 48, and 49** are rejected under 35 U.S.C. 103(a) as being unpatentable over Goodwin et al. (US 6,199,195) in view of Mullins (US 7,103,600).

As per **Claim 7**, the rejection of **Claim 1** is incorporated; however, Goodwin et al. do not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and
- spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- using reflection, reading metadata associated with the executable application to create a graph of code elements (*see Column 36: 27-40, "Now select the top level node to work with.*

Notice our use of a CocoProxyM class which 'wrappers' the pure java object model class and gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ... "); and

- spanning the graph for re-creating the model based on code elements encountered (*see Column 36: 50-67, "Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ... ".*)

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include using reflection, reading metadata associated with the executable application to create a graph of code elements; and spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements (*see Mullins – Column 2: 46-49*).

As per **Claim 8**, the rejection of **Claim 7** is incorporated; however, Goodwin et al. do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Mullins discloses:

Art Unit: 2191

- as each code element is encountered, reconstructing a corresponding portion of the model (*see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or produced from another information source UML/XMI or directly from the database schema."* and 42-50, *"This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source (*see Mullins – Column 30: 47-50*).

As per **Claim 29**, the rejection of **Claim 23** is incorporated; however, Goodwin et al. do not disclose:

- wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered (*see Column 36: 27-40, "Now select the top level node to work with. Notice our use of a CocoProxyM class which 'wrappers' the pure java object model class and gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ... "* and 50-67, *"Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ... "*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements (*see Mullins – Column 2: 46-49*).

As per **Claim 30**, the rejection of **Claim 29** is incorporated; however, Goodwin et al. do not disclose:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application.

Mullins discloses:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application (*see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or produced from another information source UML/XMI or directly from the database schema."* and 42-50, *"This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP. "*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source (*see Mullins – Column 30: 47-50*).

As per **Claim 48**, the rejection of **Claim 43** is incorporated; however, Goodwin et al. do not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and
- spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- using reflection, reading metadata associated with the executable application to create a graph of code elements (*see Column 36: 27-40, "Now select the top level node to work with.*

Notice our use of a CocoProxyM class which 'wrappers' the pure java object model class and gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ... "); and

- spanning the graph for re-creating the model based on code elements encountered (*see Column 36: 50-67, "Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ... ".*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include using reflection, reading metadata associated with the executable application to create a graph of code elements; and spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements (*see Mullins – Column 2: 46-49*).

As per **Claim 49**, the rejection of **Claim 48** is incorporated; however, Goodwin et al. do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Mullins discloses:

- as each code element is encountered, reconstructing a corresponding portion of the model (*see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or produced from another information source UML/XMI or directly from the database schema."* and 42-50, *"This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source (*see Mullins – Column 30: 47-50*).

17. **Claims 9, 31, and 50** are rejected under 35 U.S.C. 103(a) as being unpatentable over Goodwin et al. (US 6,199,195) in view of Mullins (US 7,103,600) as applied to Claims 7, 29, and 48 above, and further in view of Schloegel et al. (US 2004/0044990).

As per **Claim 9**, the rejection of **Claim 7** is incorporated; however, Goodwin et al. and Mullins do not disclose:

Art Unit: 2191

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel et al. disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques (*see Paragraph [0033]*, “*... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.*”).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel et al. into the teaching of Goodwin et al. to include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code (*see Schloegel et al. – Paragraph [0034]*).

As per **Claim 31**, the rejection of **Claim 29** is incorporated; however, Goodwin et al. and Mullins do not disclose:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel et al. disclose:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques (*see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.*").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel et al. into the teaching of Goodwin et al. to include wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code (*see Schloegel et al. – Paragraph [0034]*).

As per **Claim 50**, the rejection of **Claim 48** is incorporated; however, Goodwin et al. and Mullins do not disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel et al. disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques (*see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name,*

or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal. ").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel et al. into the teaching of Goodwin et al. to include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code (*see Schloegel et al. – Paragraph [0034]*).

18. **Claims 10, 11, 32, 33, 51, and 52** are rejected under 35 U.S.C. 103(a) as being unpatentable over Goodwin et al. (US 6,199,195) in view of Baisley (US 6,711,734).

As per **Claim 10**, the rejection of **Claim 1** is incorporated; however, Goodwin et al. do not disclose:

- detecting a class having a package element; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Baisley discloses:

- detecting a class having a package element (*see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33). "*); and

Art Unit: 2191

- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model (*see Column 4: 35-39, "If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34). "*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

As per **Claim 11**, the rejection of **Claim 10** is incorporated; however, Goodwin et al. do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- detecting an attribute specifying that a class belongs to the UML package (*see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53). "); and*
- specifying in the reconstructed model that the class belongs to that UML package (*see Column 4: 60-67 through Column 5: 1-2, "If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54).*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

As per **Claim 32**, the rejection of **Claim 23** is incorporated; however, Goodwin et al. do not disclose:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Baisley discloses:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model (*see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33). If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34). "*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include wherein the run-time framework includes submodules for detecting a class having a

package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

As per **Claim 33**, the rejection of **Claim 32** is incorporated; however, Goodwin et al. do not disclose:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package (*see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53). If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54).* ”).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

As per **Claim 51**, the rejection of **Claim 43** is incorporated; however, Goodwin et al. do not disclose:

- detecting a class having a package element; and
- creating a corresponding Unified Modeling Language (UML) package for the

reconstructed model.

Baisley discloses:

- detecting a class having a package element (*see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33). "*); and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model (*see Column 4: 35-39, "If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34).*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

As per **Claim 52**, the rejection of **Claim 51** is incorporated; however, Goodwin et al. do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- detecting an attribute specifying that a class belongs to the UML package (*see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53)."*); and

- specifying in the reconstructed model that the class belongs to that UML package (*see Column 4: 60-67 through Column 5: 1-2, "If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54)."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model (*see Baisley – Column 4: 2-8*).

19. **Claims 13, 14, 35, 36, 54, and 55** are rejected under 35 U.S.C. 103(a) as being unpatentable over Goodwin et al. (US 6,199,195) in view of Mutschler, III (US 7,162,462).

As per **Claim 13**, the rejection of **Claim 12** is incorporated; however, Goodwin et al. do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- ensuring that all classes in the model belong to a common superclass (*see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

As per **Claim 14**, the rejection of **Claim 13** is incorporated; however, Goodwin et al. do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

Mutschler, III discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes (*see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas*

representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.”).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

As per **Claim 35**, the rejection of **Claim 34** is incorporated; however, Goodwin et al. do not disclose:

- a submodule for ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- a submodule for ensuring that all classes in the model belong to a common superclass (*see Column 5: 19-21, “The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.”*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include a submodule for ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be

motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

As per **Claim 36**, the rejection of **Claim 35** is incorporated; however, Goodwin et al. do not disclose:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass.

Mutschler, III discloses:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass (*see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

As per **Claim 54**, the rejection of **Claim 53** is incorporated; however, Goodwin et al. do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- ensuring that all classes in the model belong to a common superclass (*see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived."*).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

As per **Claim 55**, the rejection of **Claim 54** is incorporated; however, Goodwin et al. do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

Mutschler, III discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes (*see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas*

representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.”).

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time (*see Mutschler, III – Column 5: 40-43*).

20. **Claim 38** is rejected under 35 U.S.C. 103(a) as being unpatentable over Goodwin et al. (US 6,199,195).

As per **Claim 38**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the model is initially created using a UML modeling tool (*see Column 8: 44-53, “Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308.” and “... a plurality of model adapters 310 for defining a translation of the logical models of the modeling tools 302, 304, 406 into unified models, expressed in a unified modeling language, such as Unified Modeling Language (UML).”*).

However, Goodwin et al. do not disclose:

- wherein the source code is compiled using a C# compiler.

Official Notice is taken that it is old and well known within the computing art to include compiling source code using a C# compiler. A compiler is an essential software component of an Integrated Development Environment (IDE) used by computer programmers to develop software, such as C# applications. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the source code is compiled using a C# compiler. The modification would be obvious because one of ordinary skill in the art would be motivated to execute applications written in C#.

Conclusion

21. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

A. Freed et al. (US 6,269,473) disclose a set of programming constructs and services that provide high levels of modularity, abstraction, dynamism and configurability.

B. Hearne et al. (US 6,701,381) disclose a client/server data processing system is developed by a method in which a meta model is transformed by a model generator into an application model, which has a layered structure reflecting that of the end-product system.

C. Barrett et al. (US 7,000,219) disclose a process for development of a flexible component-based software system, and to a system developed by such a process.

D. Kadel, Jr. et al. (US 2002/018440) disclose a framework enables data source components to be developed independently of data consumer components.

Art Unit: 2191

E. Lauterbach et al. (US 2005/0071805) disclose methods and apparatus, including computer program products, for using a metamodel as the basis for development of applications.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

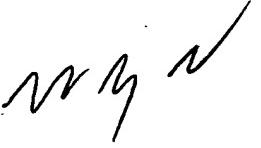
Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR

Art Unit: 2191

system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

QC / AC
January 16, 2007


WEI ZHEN
SUPERVISORY PATENT EXAMINER